

# Asymptotically Near-Optimal Motion Planning using Lower Bounds on Cost<sup>\*</sup>

Oren Salzman and Dan Halperin

Blavatnik School of Computer Science, Tel-Aviv University, Israel

**Abstract.** Asymptotically-optimal sampling-based motion-planning algorithms often, from a certain stage of their execution, invest huge computational resources at only slightly improving the cost of the current best existing solution. We aim to overcome this problem by (i) carefully using lower bounds on the cost between configurations and by (ii) relaxing asymptotic optimality to asymptotic near-optimality. To this end, we present Motion Planning using Lower Bounds (MPLB), an anytime asymptotic near-optimal algorithm, based on the Fast Marching Trees (FMT\*) algorithm [15]. An advantageous by-product of our approach is that we can reach situations where the weight of collision detection is almost negligible with respect to nearest-neighbor calls; this is in the spirit of a recent suggestion by Bialkowski et al. [6]. We prove that MPLB performs **no more** collision-detection calls than an anytime version of FMT\* (called aFMT\*) and bound the additional number of nearest-neighbor calls for a family of configuration spaces. We show experimentally that for certain scenarios, MPLB produces lower cost paths faster than aFMT\*, spending less than 20% of its time on collision detection.

**Keywords:** Motion Planning, Lower Bounds, Collision Detection

## 1 Introduction

Motion-planning algorithms aim to find a collision-free path for a robot moving amidst obstacles. The general problem is PSPACE-hard when the number of degrees of freedom (DoF) is part of the input [25]. Thus, the most prevalent approach in practice is to use sampling-based techniques and relaxing completeness to probabilistically completeness [8]. These algorithms sample points in the robot’s *configuration-space* (C-space) and connect close-by configurations to construct a graph called a *roadmap*. Often, a *low-cost* path is desired, where path cost can be measured in terms of, for example, length, clearance or energy consumption along the path.

---

<sup>\*</sup> This work has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL—Computational Geometry Learning), by the Israel Science Foundation (grant no. 1102/11), by the German-Israeli Foundation (grant no. 1150-82.6/2011), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

Sampling-based algorithms rely on two central primitive operations: *Collision Detection* (CD) and *Nearest Neighbors* (NN) search. CD determines whether a configuration is collision-free or not and is often used to assess if a path connecting close-by configurations is collision-free. This latter use is referred to as *Local Planning* (LP). An NN data structure preprocesses a set of points to efficiently answer queries such as “which is the closest point” or “which are the points within radius  $r$ ” of a query point? In practice, the cost of CD, primarily due to LP calls, often dominates the running time of sampling-based algorithms, and is typically regarded as the computational bottleneck for such algorithms. For a summary of the computational complexity of NN, CD and LP in sampling-based motion-planning algorithms see [6].

Although sampling-based algorithms have appeared in the literature since the mid 90’s (see, e.g., [12, 17, 20] to mention just a few), only recently an algorithm that has guarantees on the cost of the produced path has been suggested. In their influential work, Karaman and Frazzoli [16] analyzed existing sampling-based algorithms (namely PRM [17] and RRT [20]) and introduced the notion of *asymptotic optimality* (AO); an algorithm is said to be AO if the cost of the solution produced by it converges to the cost of the optimal solution if the algorithm is run for sufficiently long time. They proposed AO variants of PRM and RRT called PRM\* and RRT\*, respectively. In these latter variants, each node in the roadmap should consider connections to all nodes within a neighborhood of radius proportional to  $\left(\frac{\log n}{n}\right)^{\frac{1}{d}}$ , where  $n$  is the number of collision-free samples used by the algorithm and  $d$  is the dimension of the C-space. However, the AO of the PRM\* and RRT\* algorithms comes at the cost of increased running time and memory consumption when compared to their non-optimal counterparts. To reduce this cost, several improvements were proposed which modify the sampling scheme [2, 14], the collision detection [6], or relax the optimality to *asymptotic near-optimality* (ANO) [10, 22, 23, 26]. An algorithm is said to be ANO if, given an approximation factor  $\varepsilon$ , the cost of a solution returned by the algorithm is guaranteed to converge to within a factor of  $1 + \varepsilon$  of the cost of the optimal (minimal) solution.

Following the introduction of PRM\* and RRT\*, other AO algorithms have been suggested. The first, by Arslan and Tsiotras [4] borrowed ideas used from the Lifelong Planning A\* algorithm [18]. They suggest RRT# (RRT sharp), which also guarantees AO, but, in addition, ensures that the constructed spanning tree rooted at the initial state contains lowest-cost path information for vertices which have the potential to be part of the optimal solution. RRT# extends its roadmap in a similar fashion to RRT\* but adds a replanning procedure. Thus, in contrast to RRT\* which only performs *local* rewiring of the search tree, RRT# efficiently propagates changes to *all* the relevant parts of the roadmap.

Another AO algorithm, proposed by Janson and Pavone, is the *Fast Marching Trees* (FMT\*) [15] algorithm. FMT\* is shown to converge to an optimal solution faster than PRM\* or RRT\*. It uses a set of probabilistically-drawn configurations to construct a tree, which grows in cost-to-come space (see Section 2 for more details).

We wish to work in the *anytime* setting, i.e., to consider algorithms that can return effective results even within a limited time budget, which is not known a priori. Such algorithms typically find a solution quickly and refine it as time permits. Hence, we first introduce aFMT\*—an anytime variant of FMT\* (see Section 3). Now, looking into the anytime AO sampling-based algorithms (namely RRT\*, RRT# and aFMT\*), one can see that they are incremental in the following sense: They attempt to improve the cost of the existing solution regardless of the amount of improvement that may be achieved. This may incur substantial computational overhead with infinitesimal improvements.

**Contribution and paper organization** In this work we concentrate on path length as the cost measure. We show that by introducing an *approximation factor*  $\varepsilon$ , and using Euclidean distances as *effective lower bounds* on the cost of edges we ensure that calls to the expensive local planner will occur only for nodes that have the potential to improve the solution at hand by a large factor. We call our scheme Motion Planning using Lower Bounds or MPLB for short. The implications of the scheme are twofold: First, for certain scenarios, our ANO algorithm explores the C-space faster than its AO counterpart and finds paths of lower cost much faster. This resembles the work by Alterovitch et al. [3] who propose a lazy variant of RRT\* to decrease the number of collision checks while still retaining optimality guarantees. Their work, though, relies on a user-provided parameter and a general scheme for selecting this parameter is not specified. In contrast, the only parameter used by our algorithm is  $\varepsilon$ .

The second implication, which is possibly more interesting, is that after applying these algorithmic tools, the weight of CD and LP becomes negligible when compared to that of the NN calls<sup>1</sup>. Bialkowski et al. [6] suggested to replace CD calls by NN calls. Their scheme relies on additional data produced by the CD algorithm used, namely, a bound on the clearance of a configuration (if it is collision free) or on its penetration depth (if it is not collision free). We note that such a bound, which should be computed in the *C-space* is not trivial to compute for some prevalent spaces. In contrast, MPLB can use *any* NN, CD and LP procedure. Thus, our results, which are more general as they are applicable to general C-spaces, strengthen the conjecture of Bialkowski et al. [6] that NN computation and not CD is the bottleneck of motion planning algorithms.

This work continues and expands our recent work [26] where we relaxed the AO of RRT\* to ANO using lower bounds. The novel component here is that multiple nodes are processed *simultaneously* while in the RRT\* algorithm the nodes are processed one at a time. The difference allows to further exploit lower bounds when relaxing AO to ANO. Specifically, by using NN calls, the algorithm is able to identify where there is no need to perform LP calls, hence dramatically reducing the computation time.

Moreover, if we know the number  $n$  of samples used by our algorithm, we are only interested in two types of NN calls: Finding the NN of a node and finding

<sup>1</sup> Throughout this paper, when we say a NN call we mean a call to find all the nodes within radius  $r(n)$  of a given node.  $r(n)$  is a radius depending on the number of samples used and will be formally defined in Section 2.

all the samples within a radius  $r(n)$  of a given node. (Alternatively, certain sampling-based algorithms use a procedure to find the  $k$  NN of a node for some parameter  $k$ .) This suggests it would be desirable to devise application-specific data structures that are tailored to solve such queries only and are faster than existing, general purpose NN data structures.

Our framework is demonstrated via the FMT\* algorithm which is reviewed in Section 2. We introduce a straightforward adaptation of FMT\* for anytime planning which we call aFMT\* in Section 3 and then proceed to present our algorithm MPLB in Section 4. We analyze aFMT\* and MPLB with respect to the amount of calls to the NN and LP procedures in Section 5 and show that MPLB performs no more LP calls than aFMT\*. This potentially results in much fewer CD calls. Moreover, we prove bounds on the additional number of NN computations for a family of C-spaces. We continue to present experimental results in Section 6 and conclude with a discussion and suggestions for future work in Section 7.

## 2 Terminology and algorithmic background

We begin by formally stating the motion-planning problem and introducing several standard procedures used by sampling-based algorithms. We continue by reviewing the FMT\* algorithm.

### 2.1 Problem definition and terminology

Let  $\mathcal{X}$ ,  $\mathcal{X}_{\text{free}}$  denote the Euclidean<sup>2</sup> C-space and free space, respectively, and  $d$  the dimension of the C-space. Let  $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$  be the motion-planning problem where:  $x_{\text{init}} \in \mathcal{X}_{\text{free}}$  is the initial free configuration of the robot and  $\mathcal{X}_{\text{goal}} \subseteq \mathcal{X}_{\text{free}}$  is the goal region. We will make use of the following procedures: **sample.free**( $n$ ), a procedure returning  $n$  random configurations from  $\mathcal{X}_{\text{free}}$ ; **nearest\_neighbors**( $x, V, r$ ) is a procedure that returns all neighbors of  $x$  with distance smaller than  $r$  within the set  $V$ ; **collision\_free**( $x, y$ ) tests whether the straight-line segment connecting  $x$  and  $y$  is contained in  $\mathcal{X}_{\text{free}}$ ; **cost**( $x, y$ ) returns the cost of the straight-line path connecting  $x$  and  $y$ , namely, in our case, the distance. We consider weighted graphs  $\mathcal{G} = (V, E)$ , where the weight of an edge  $(x, y) \in E$  is **cost**( $x, y$ ). Given such a graph  $\mathcal{G}$ , we denote by **cost** $_{\mathcal{G}}$ ( $x, y$ ) the weighted shortest path from  $x$  to  $y$ . Let **cost-to-come** $_{\mathcal{G}}$ ( $x$ ) be **cost** $_{\mathcal{G}}$ ( $x_{\text{init}}, x$ ) and **cost-to-go** $_{\mathcal{G}}$ ( $x$ ) be the minimal **cost** $_{\mathcal{G}}$ ( $x, x_{\text{goal}}$ ) for  $x_{\text{goal}} \in \mathcal{X}_{\text{goal}}$ . Namely for every node  $x$ , **cost-to-come** $_{\mathcal{G}}$ ( $x$ ) and **cost-to-go** $_{\mathcal{G}}$ ( $x$ ) is the minimal cost to reach  $x$  from  $x_{\text{init}}$  and  $\mathcal{X}_{\text{goal}}$ , respectively. Additionally, let  $B_{\mathcal{G}}(x_{\text{init}}, r)$ ,  $B_{\mathcal{G}}(\mathcal{X}_{\text{goal}}, r)$  be the set of all nodes whose cost-to-come (respectively, cost-to-go) value on  $\mathcal{G}$  is smaller than  $r$ .

<sup>2</sup> Although we describe the algorithm for Euclidean spaces, by standard techniques (see, e.g. [8, Section 3.5, Section 7.1.2], [21, Chapters 4-5] or [19]) the algorithm can be applied to non-Euclidean spaces such as SE3.

Given a set of samples  $V$ , and a radius  $r$ , we denote by  $G(V, r)$  the *disk graph*<sup>3</sup> which is the graph whose set of vertices is  $V$  and two vertices  $x, y$  are connected by an edge if the distance between  $x$  and  $y$  is less than  $r$ .

## 2.2 Fast Marching Trees (FMT\*)

The FMT\* algorithm [15] performs a “lazy” dynamic programming recursion on a set of sampled configurations to grow a tree rooted at  $x_{init}$ . The algorithm grows the tree steadily outward in cost-to-come space. The pseudo-code of FMT\* is outlined in Algorithm 1.

The FMT\* algorithm samples  $n$  collision-free nodes  $V$  (line 1) and builds a minimum-cost spanning tree rooted at the initial configuration by maintaining two sets of nodes  $H, W$  such that  $H$  is the set of nodes added to the tree that may be expanded and  $W$  is the set of nodes that have not been expanded yet (line 2). It then computes for each node the set of nearest neighbors<sup>4</sup> of radius  $r(n)$  (line 4). The algorithm repeats the following process: the node  $z$  with the lowest cost-to-come value is chosen from  $H$  (line 5 and 19). For each neighbor  $x$  of  $z$  that is not already in  $H$ , the algorithm finds its neighbor  $y \in H$  such that the cost-to-come of  $y$  added to the distance between  $y$  and  $x$  is minimal (lines 8-11). If the local path between  $y$  and  $x$  is free,  $x$  is added to  $H$  with  $y$  as its parent (lines 13-15). At the end of each iteration  $z$  is removed from  $H$  (line 16). The algorithm runs until a solution is found or there are no more nodes to process.

The radius used by the algorithm is

$$r(n) = (1 + \eta) \cdot 2 \left( \frac{1}{d} \right)^{\frac{1}{d}} \left( \frac{\mu(\mathcal{X}_{\text{free}})}{\zeta_d} \right) \left( \frac{\log n}{n} \right)^{\frac{1}{d}}, \quad (1)$$

where  $\eta > 0$  is some constant,  $\mu(\cdot)$  denotes the  $d$ -dimensional Lebesgue measure and  $\zeta_d$  is the volume of the unit ball in the  $d$ -dimensional Euclidean space. This value is smaller than the radius used by Karaman and Frazzoli [16] due to the different definition of AO used by Janson and Pavone. Specifically, Karaman and Frazzoli [16] use the notion of *convergence almost everywhere* while Janson and Pavone [15] use the (weaker) notion of *convergence in probability*.

## 3 Anytime FMT\* (aFMT\*)

An algorithm is said to be *anytime* if it yields meaningful results even after a short time and it improves the quality of the solution as more computation time is available. We outline a straightforward enhancement to FMT\* to make it anytime. As noted in previous work (see, e.g., [28]) one can turn a batch algorithm into an anytime one by the following (general) approach: choose an

<sup>3</sup> The disk graph is sometimes referred to as the *Ball graph* or the *Neighborhood graph*.

<sup>4</sup> The nearest-neighbor computation can be delayed and performed only when needed but we present the batched mode of computation to simplify the exposition.

---

**Algorithm 1** fast\_marching\_tree

---

```
1:  $V \leftarrow \{x_{\text{init}}\} \cup \text{sample\_free}(n); \quad E \leftarrow \emptyset; \quad G \leftarrow (V, E)$ 
2:  $W \leftarrow V \setminus \{x_{\text{init}}\}; \quad H \leftarrow \{x_{\text{init}}\}$ 
3: for all  $v \in V$  do
4:    $N_v \leftarrow \text{nearest\_neighbors}(V \setminus \{v\}, v, r(n))$ 
5:  $z \leftarrow x_{\text{init}}$ 
6: while  $z \notin \mathcal{X}_{\text{Goal}}$  do
7:    $H_{\text{new}} \leftarrow \emptyset$ 
8:    $X_{\text{near}} \leftarrow W \cap N_z$  // nearest vertices of  $z$  not yet added to the roadmap tree
9:   for  $x \in X_{\text{near}}$  do
10:     $Y_{\text{near}} \leftarrow H \cap N_x$  // nearest vertices of  $x$  already added to the roadmap tree
11:     $y_{\min} \leftarrow \arg \min_{y \in Y_{\text{near}}} \{\text{cost-to-come}_G(y) + \text{cost}(y, x)\}$ 
12:    if  $\text{collision\_free}(y_{\min}, x)$  then
13:       $\text{parent}(x) \leftarrow y_{\min}$ 
14:       $H_{\text{new}} \leftarrow H_{\text{new}} \cup \{x\}$ 
15:       $W \leftarrow W \setminus \{x\}$ 
16:     $H \leftarrow (H \cup H_{\text{new}}) \setminus \{z\}$ 
17:    if  $H = \emptyset$  then
18:      return FAILURE
19:     $z \leftarrow \arg \min_{y \in H} \{\text{cost-to-come}_G(y)\}$ 
20: return PATH // the path is retrieved by following the parent pointers
```

---

initial (small) number of samples  $n = n_0$  and apply the algorithm. As long as time permits, double  $n$  and repeat the process. The total running time is less than twice that of the running time for the largest  $n$  and the asymptotic running time is not worse than if we knew the largest value for  $n$  in advance. Note that as FMT\* is AO, aFMT\* is AO as well.

We can further speed up this method by reusing both existing samples and connections from previous iterations. Notice that this improvement does not change the asymptotic running time. It affects only the constants of the running time. Assume the algorithm was run with  $n$  samples and now we wish to re-run it with  $2n$  samples. In order to obtain the  $2n$  random samples, we take the  $n$  random samples from the previous iteration together with  $n$  new additional random samples. For each node that was used in iteration  $i - 1$ , around half of its neighbors in iteration  $i$  are nodes from iteration  $i - 1$  and half of its neighbors are newly-sampled nodes. Thus, if we save the results of calls to the local planner, we can cache them to be reused in future iterations. For more information on this caching, see Appendix B.

## 4 Algorithmic framework

Given a random infinite sequence of collision-free samples  $S = s_1, s_2, \dots$  denote by  $V_i(S)$  the set of the first  $2^i$  elements of  $S$ . Let  $\mathcal{G}_i(S) = G(V_i(S), r(|V_i(S)|))$  and let  $\mathcal{H}_i(S) \subseteq \mathcal{G}_i(S)$  be the subgraph containing collision-free edges only (here  $r(n)$  is the radius defined in Eq. 1). For brevity, we omit  $S$  when referring

to  $V_i(S)$ ,  $\mathcal{G}_i(S)$  and  $\mathcal{H}_i(S)$ . Moreover, when we compare our algorithm to the aFMT\* algorithm, we do so for runs on the same random infinite sequence  $S$ . Clearly, for any two nodes  $x, y \in V_i$ ,  $\text{cost}_{\mathcal{G}_i}(x, y) \leq \text{cost}_{\mathcal{H}_i}(x, y)$ . Thus for any node  $x \in V_i$ ,  $\text{cost-to-go}_{\mathcal{G}_i}(x) \leq \text{cost-to-go}_{\mathcal{H}_i}(x)$ . Namely, the cost-to-go computed using the disk graph  $\mathcal{G}_i$  is a lower bound on the cost-to-go that may be obtained using  $\mathcal{H}_i$ . We call this the **lower bound property**.

We present Motion Planning using Lower Bounds, or MPLB. Similar to aFMT\*, the algorithm runs in iterations and at the  $i$ 'th iteration, uses  $V_i$  as its set of samples. Unlike aFMT\*, each iteration consists of a **preprocessing phase** of computing a lower bound on the cost-to-go values and a **searching phase** where a modified version of aFMT\* is used.

Let  $c_i(\text{ALG})$  denote the cost of the solution obtained by an algorithm ALG using  $V_i$  as the set of samples (set  $c_0(\text{MPLB}) \leftarrow \infty$ ). Given an *approximation factor*  $\varepsilon$ , MPLB maintains the following invariant:

$$c_i(\text{aFMT}^*) \leq c_i(\text{MPLB}) \leq (1 + \varepsilon) \cdot c_i(\text{aFMT}^*).$$

Namely, at each iteration, the cost of the solution obtained by MPLB is within a factor of  $1 + \varepsilon$  from the solution that aFMT\* would obtain for the same set of samples. We call this the **bounded approximation invariant**. As aFMT\* is AO, and if the bounded approximation invariant is indeed maintained then the following holds.

**Corollary 1** *The MPLB algorithm is asymptotically near-optimal.*

We give a formal proof that the bounded approximation invariant is maintained in Section 4.4.

We now show that only a subset of the nodes sampled in each iteration should be considered. We then proceed to describe the preprocessing phase and the searching phase of MPLB. In Appendix A we provide a graphic demonstration comparing the behavior of MPLB and aFMT\* in different iterations. This can serve as an illustrated accompaniment to the verbal description here.

#### 4.1 Promising nodes

We use the lower bound property to consider only a *subset* of  $V_i$  that will be used in the  $i$ 'th iteration. Intuitively, we only wish to consider nodes that may produce a solution that is significantly (by more than a factor of  $(1 + \varepsilon)$ ) better than the solution produced in previous iterations. This leads us to the definition of promising nodes:

**Definition 1.** *A node  $x \in V_i$  is **promising** (at iteration  $i$ ) if*

$$\text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost-to-go}_{\mathcal{H}_i}(x) < \frac{c_{i-1}(\text{MPLB})}{1 + \varepsilon}.$$

Assume that the bounded approximation invariant was maintained for the  $i - 1$ 'st iteration. Using the notion of promising nodes we can identify when the solution produced by MPLB in the  $i - 1$ 'st iteration does not violate the bounded approximation invariant for the  $i$ 'th iteration.

**Lemma 1** *If the solution produced by aFMT\* in the  $i$ 'th iteration contains a node that is not promising then,  $c_{i-1}(\text{MPLB}) \leq (1 + \varepsilon) \cdot c_i(\text{aFMT}^*)$ .*

*Proof.* Clearly, any node  $x$  that is on the optimal path to  $\mathcal{X}_{\text{goal}}$  produced by aFMT\* in the  $i$ 'th iteration has  $\text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost-to-go}_{\mathcal{H}_i}(x) = c_i(\text{aFMT}^*)$ . Now, let  $x$  be a node that is *not* promising (at iteration  $i$ ), thus  $\text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost-to-go}_{\mathcal{H}_i}(x) \geq \frac{c_{i-1}(\text{MPLB})}{1+\varepsilon}$ . If  $x$  lies on the optimal path produced by aFMT\* in the  $i$ 'th iteration, then  $c_i(\text{aFMT}^*) \geq \frac{c_{i-1}(\text{MPLB})}{1+\varepsilon}$ . Thus, the solution produced in the  $i - 1$ 'st iteration by MPLB does not violate the bounded approximation invariant in the  $i$ 'th iteration.  $\square$

In the preprocessing phase, MPLB will traverse  $\mathcal{G}_i$  (and not  $\mathcal{H}_i$ ) to collect a set of nodes that contains all promising nodes (and possibly other nodes), compute a lower bound on their cost-to-go and use this set in the searching phase.

## 4.2 Preprocessing phase: Estimating the cost-to-go

Recall that in the preprocessing phase, we wish to compute a lower bound on the cost-to-go for (a subset of) nodes  $x \in V_i$ . Specifically, the only nodes we wish to consider are *promising nodes*. This is done by collecting the set of nodes  $V_{\text{preproc}} = B_{\mathcal{G}_i}\left(x_{\text{init}}, \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)}\right) \cup B_{\mathcal{G}_i}\left(\mathcal{X}_{\text{goal}}, \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)}\right)$ . Namely, by performing one traversal from  $x_{\text{init}}$  and one traversal from  $\mathcal{X}_{\text{goal}}$ , all nodes such that  $\text{cost-to-come}_{\mathcal{G}_i} \leq \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)}$  or  $\text{cost-to-go}_{\mathcal{G}_i} \leq \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)}$  are found. Clearly, any node *not* in either set is not promising.

After collecting all nodes in  $V_{\text{preproc}}$ , MPLB computes the distance of every such node from  $\mathcal{X}_{\text{goal}}$ . This is done by running a shortest paths algorithm on the graph  $\mathcal{G}_i$  restricted to the nodes in  $V_{\text{preproc}}$ . This distance is stored for each node and will be used as a lower bound on the cost-to-go. We note that this preprocessing phase is implemented by three traversals over (a subset of)  $V_i$  and only uses NN calls and no CD calls (as there are no LP calls).

## 4.3 Searching phase: Using cost-to-go estimations

The lower bounds computed allow for two algorithmic enhancements to the searching phase when compared to aFMT\*: **(i)** Speeding up the running time of the algorithm by incorporating the cost-to-go estimation in the ordering scheme of the nodes and **(ii)** discarding nodes that are found to be not promising.

**Node ordering:** Recall that in the aFMT\* algorithm,  $H$  is the set of nodes added to the tree that may be expanded and that these nodes are ordered according to their cost-to-come value (Algorithm 1, line 19). Instead, we suggest using the cost-to-come added to the cost-to-go estimation to order the nodes in  $H$ . This follows exactly the formulation of the A\* algorithm [24]: Recall that A\* performs a Dijkstra-like search on a set of nodes. The nodes that were encountered but not processed yet ( $H$  in our setting) are ordered according to a cost function  $f() = g() + h()$ . Here,  $g(x)$  is the (computed) cost-to-come value of



$x$  ( $\text{cost-to-come}_{\mathcal{H}_i}(x)$  in our case) and  $h$  is a lower bound on the cost-to-go of  $x$  to the goal ( $\text{cost-to-go}_{\mathcal{G}_i}(x)$  in our case). FMT\* essentially uses  $h = 0$  which is indeed a (trivial) admissible heuristic. Instead, we suggest to use a much sharper bound in order to speed up the search towards the goal.

**Discarding nodes:** In the preprocessing stage MPLB computes a set of nodes that *may* be promising. For each such node, the cost-to-come value was not computed. In the searching phase, each node added to the tree has a cost-to-come value that will not change in the current iteration. Thus, for every node  $x$  added to the tree with  $\text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost-to-go}_{\mathcal{G}_i}(x) \geq \frac{c_{i-1}(\text{MPLB})}{1+\varepsilon}$ ,  $x$  may be discarded as it cannot be promising. Hence, the algorithm will terminate an iteration when it is evident that the solution obtained in the previous iteration cannot be improved by a factor of more than  $1 + \varepsilon$ .

We note that (i) there may be iterations where none of the nodes may be promising and the searching phase will terminate immediately and that (ii) as we will see in the sequel, using lower bounds has a significant effect on the running time of the algorithm in practice.

#### 4.4 Asymptotic near-optimality of MPLB

In order to show that MPLB is ANO we first show that,

**Lemma 2** *The bounded approximation invariant is maintained by MPLB.*

*Proof.* We prove the lemma by induction over the number of iterations.

**Induction base:** At the first iteration,  $c_0(\text{MPLB}) \leftarrow \infty$  and all nodes are promising. Thus, both the aFMT\* algorithm and the MPLB algorithm use the same set of nodes  $V_1$ . The difference between the two algorithm are (i) the order by which nodes are processed in the searching phase and (ii) the possible discarding of nodes.

Note the following observation, which follows directly from the optimality proof of the A\* algorithm using any admissible heuristic.

**Observation 1** *Given a fixed set of nodes, namely if no nodes are discarded, then both the old ordering scheme (using only cost-to-come) and the new ordering scheme (using cost-to-come+cost-to-go) will return the same path.*

We show that if  $x_{\text{init}}$  is in the same connected component than any node in  $\mathcal{X}_{\text{goal}}$  then no nodes are discarded (if not, then neither aFMT\* nor MPLB can return a path to  $\mathcal{X}_{\text{goal}}$ ): Assume falsely that a node  $x$  is discarded, thus  $\text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost-to-go}_{\mathcal{G}_i}(x) \geq \frac{c_{i-1}(\text{MPLB})}{1+\varepsilon} = \infty$ . As  $\text{cost-to-come}_{\mathcal{H}_i}(x)$  is bounded (and thus  $x$  is in the same connected component as  $x_{\text{init}}$ ) we have that  $\text{cost-to-go}_{\mathcal{G}_i}(x) = \infty$ . This implies that  $x$  is in a different connected component than any node in  $\mathcal{X}_{\text{goal}}$  in contradiction to our assumption.

**Induction step:** Assume that the optimal path produced by aFMT\* in the  $i$ 'th iteration contains only promising nodes (if not, by Lemma 1, the bounded approximation invariant is maintained trivially). Clearly, all the promising nodes

are members of the set  $V_{\text{preproc}}$  computed by MPLB in the preprocessing stage and none of them will be discarded. Thus, by Observation 1,  $c_i(\text{MPLB}) = c_i(\text{aFMT}^*)$  and the approximation invariant is maintained.  $\square$

Using the AO of aFMT\* and Lemma 2, Corollary 1 immediately follows.

## 5 Comparative analysis

We compare the aFMT\* and MPLB algorithms with respect to the size of the tree constructed in the searching phase and with respect to the primitive procedures, namely NN and LP. We assume that the computational cost of the graph traversal algorithms is negligible.

We start by quantifying the number of NN and LP calls performed by both algorithms for the general setting where no assumptions on the C-space are made. We show that MPLB will perform *no more* calls to the local planner than aFMT\*. It *may* perform *more* nearest-neighbors calls than aFMT\*. The analysis of the number of LP calls is purely combinatorial. For any fixed sequence  $S$  of collision-free samples we show that any LP call that is made by MPLB is also made by aFMT\*.

As the NN calls take the lion's share of the computation (as will be demonstrated in Section 6), we continue our analysis by focusing on the number of NN calls performed by the algorithms. This analysis is probabilistic and is performed over the probability space defined over all sequences of random samples. We analyze the simple case where the C-space is Euclidean and contains no obstacles and show that as the number of samples tends to infinity, with high probability, MPLB will perform *less* NN calls than aFMT\*. Using the same techniques we also investigate C-spaces with obstacles that are **well behaved** (see Section 5.3 for a formal definition) and in these settings, we bound the additional number of NN calls that MPLB may perform when compared to aFMT\*. The analysis is performed for Euclidean spaces though it can be generalized to non-Euclidean C-space such as SE3 (see Footnote 2).

### 5.1 General setting

We denote by  $\#_{\text{NN},i}(\text{ALG})$ ,  $\#_{\text{LP},i}(\text{ALG})$  the number of NN and LP calls performed by an algorithm ALG in iteration  $i$ , respectively. When comparing these values for two different algorithms, we consider the same sequence  $S$  of samples.

**Search-tree size:** Let  $V_i(\text{ALG})$  denote the set of nodes in the tree in the  $i$ 'th iteration of an algorithm ALG.

**Lemma 3** *The set of nodes traversed at every iteration of the searching phase by the MPLB algorithm is not larger than that of aFMT\*.*

*Proof.* Every node  $x$  in the tree of  $\text{aFMT}^*$  has  $\text{cost-to-come}$  not larger than  $c_i(\text{aFMT}^*)$ . Thus, the size of the search-tree of  $\text{aFMT}^*$  is:  $|V_i(\text{aFMT}^*)| = |\{x \in V_i \mid \text{cost-to-come}_{\mathcal{H}_i}(x) \leq c_i(\text{aFMT}^*)\}|$ .

Similar to  $\text{aFMT}^*$  (see Observation 1), each node  $x$  traversed by MPLB in the searching phase has  $\text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost-to-go}_{\mathcal{G}_i}(x) \leq c_i(\text{aFMT}^*)$ . Additionally, due to node discarding (see Section 4),  $\text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost-to-go}_{\mathcal{G}_i}(x) \leq \frac{c_{i-1}(\text{MPLB})}{1+\varepsilon}$ . Thus, the size of the search-tree of MPLB is:  $|V_i(\text{MPLB})| = \left| \left\{ x \in V_i \mid \text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost-to-go}_{\mathcal{G}_i}(x) \leq \min \left\{ \frac{c_{i-1}(\text{MPLB})}{1+\varepsilon}, c_i(\text{aFMT}^*) \right\} \right\} \right|$ . Namely,  $|V_i(\text{MPLB})| \leq |V_i(\text{aFMT}^*)|$ .  $\square$

**Nearest neighbor calls (NN):** For every node in the search tree (of either  $\text{aFMT}^*$  or MPLB), there is an NN call (see line 8 in Algorithm 1). There may be additional NN calls for nodes that are neighbors of nodes in the search tree (see line 10 in Algorithm 1). Thus,

**Observation 2** *The number of NN calls performed by  $\text{aFMT}^*$  can be bounded from below as follows:  $\#_{\text{NN},i}(\text{aFMT}^*) \geq |V_i(\text{aFMT}^*)|$ .*

The MPLB algorithm has additional NN calls due to the preprocessing stage. More specifically, for each node traversed in the preprocessing phase, there is one NN call. Note that in the searching phase, MPLB uses only nodes traversed in the preprocessing phase, hence, there will be no additional NN calls in the searching phase. Thus,

**Observation 3** *The number of NN calls performed by MPLB is:*  
 $\#_{\text{NN},i}(\text{MPLB}) = \left| \left\{ x \in V_i \mid x \in B_{\mathcal{G}_i} \left( x_{\text{init}}, \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)} \right) \cup B_{\mathcal{G}_i} \left( \mathcal{X}_{\text{goal}}, \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)} \right) \right\} \right|$ .

**Local planning calls (LP):** Recall that the local planner is a procedure that tests if the straight-line segment connecting two configurations is collision-free. The number of LP calls depends on the C-space and is somewhat harder to quantify. Yet, the LP procedure will be called whenever either algorithm ( $\text{aFMT}^*$  or MPLB) attempts to insert a node to the search-tree (line 12 in Algorithm 1). Thus we can state the following lemma:

**Lemma 4** *If MPLB performs an LP call for the edge  $(x, y)$  in the  $i$ 'th iteration then  $\text{aFMT}^*$  will perform an LP call for the edge  $(x, y)$  as well.*

*Proof.* The LP procedure will be called for every pair of nodes  $x, y$  in the search tree such that: (i)  $x, y$  are neighbors in  $\mathcal{G}_i$  (namely their distance is less than  $r(|V_i|)$ ), (ii)  $\text{cost-to-come}_{\mathcal{H}_i}(x) < \text{cost-to-come}_{\mathcal{H}_i}(y)$  (namely  $x$  is inserted to the tree before  $y$ ), and (iii) the edge  $(z, y)$  is not collision-free for every  $z$  that is a neighbor of  $y$  in  $\mathcal{G}_i$  and  $\text{cost-to-come}_{\mathcal{H}_i}(z) + \text{cost}(z, y) \leq \text{cost-to-come}_{\mathcal{H}_i}(x) + \text{cost}(z, x)$  (namely all other neighbors  $z$  of  $y$  in the tree that could potentially lead to smaller cost-to-come values of  $y$ ).

If MPLB performs an LP call for the edge  $(x, y)$  in the  $i$ 'th iteration then conditions (i),(ii) and (iii) hold for the samples  $x, y$  in MPLB. To prove the

lemma we show that they hold for the samples  $x, y$  in aFMT\*. Condition (i) holds trivially as it is a property of the samples. Note that the cost-to-come of any node  $z$  computed by both algorithms equals to  $\text{cost-to-come}_{\mathcal{H}_i}(z)$ . Using this observation and the fact that  $V_{\text{preproc}} \subseteq V_i$  (namely the nodes used by MPLB is a subset of the nodes used by aFMT\*), conditions (ii) and (iii) hold as well.  $\square$

From the above analysis we can conclude that MPLB will perform **no more** LP calls than aFMT\*. It **may** perform **more** NN calls than aFMT\*.

## 5.2 The number of NN calls for the case of no obstacles

For preliminary exposition we assume a very simple setting where the C-space is the interior of a unit hypercube (axis-aligned box of edge length 1) in  $\mathbb{R}^d$ . In spite of the simplicity of this setting we believe that this is an informative etude—it tells us how a planner performs in “easy” portions of the C-space.

For this case, where  $\mathcal{X}_{\text{free}} = \mathcal{X}$ , we show that with high probability (w.h.p.), the number of NN calls performed by aFMT\* is larger than the number of NN calls performed by MPLB. To further simplify the exposition, we assume that the goal region consists of a single point, i.e.,  $\mathcal{X}_{\text{goal}} = \{x_{\text{goal}}\}$  (which is added to the set of samples used). Moreover, for simplicity, our analysis ignores boundary conditions where nodes are close to the bounding box of the C-space.

Recall that  $S = s_1, s_2 \dots$  is an infinite sequence of collision-free samples chosen independently and uniformly at random, and  $V_i(S)$  is the set containing the first  $2^i$  samples in  $S$ . We further assume that each sample is chosen independently and uniformly at random inside the hypercube  $[0, 1]^d$ . We denote by  $X_{\text{NN},i}(\text{ALG}(S))$  the random variable counting the number of NN calls performed by an algorithm ALG in iteration  $i$  run with  $S$  as its set of samples. As this section contains probabilistic arguments, as opposed to Section 4 and to the analysis presented in Section 5.1 we explicitly denote  $S$  for all parameters depending on  $S$ .

**Theorem 1** *There is an integer  $I_0 > 0$  such that for every  $i > I_0$ , w.h.p.,*

$$X_{\text{NN},i}(\text{MPLB}(S)) < X_{\text{NN},i}(\text{aFMT}^*(S)).$$

*Proof.* We bound  $X_{\text{NN},i}(\text{MPLB}(S))$  from above and bound  $X_{\text{NN},i}(\text{aFMT}^*(S))$  from below w.h.p. and show that the upper bound on  $X_{\text{NN},i}(\text{MPLB}(S))$  is smaller than the lower bound on  $X_{\text{NN},i}(\text{aFMT}^*(S))$ . The proof uses two main components: (i) a bound on  $\text{cost}_{\mathcal{H}_i(S)}(x, y)$  with respect to the Euclidean distance between any two nodes  $x$  and  $y$  and (ii) a bound on the number of samples within a ball located in  $\mathcal{X}_{\text{free}}$ .

**Lemma 5** *If  $\mathcal{X} = \mathcal{X}_{\text{free}}$ , for a sufficiently large  $i$ ,  $\forall x, y \in V_i(S)$  and  $\forall \lambda > 0$ , w.h.p.,  $\text{cost}(x, y) \leq \text{cost}_{\mathcal{H}_i(S)}(x, y) \leq (1 + \lambda)\text{cost}(x, y)$ .*

*Proof.* Recall that the minimal path length computed between any two vertices of a roadmap is an upper bound on the Euclidean distance between the vertices, thus, for any  $S$ ,  $\text{cost}(x, y) \leq \text{cost}_{\mathcal{H}_i(S)}(x, y)$ . Now, the proof of AO of the FMT\* algorithm (Theorem 2 in [15]) ensures that for all  $\lambda > 0$ ,

$$\lim_{i \rightarrow \infty} \Pr[\text{cost}_{\mathcal{H}_i(S)}(x, y) > (1 + \lambda)\text{cost}(\pi^*(x, y))] = 0 \quad \forall x, y \in V_i(S),$$

where  $\pi^*(x, y)$  is the shortest path in  $\mathcal{X}_{\text{free}}$  between  $x$  and  $y$  and  $\text{cost}(\pi^*(x, y))$  is its length. For the case that  $\mathcal{X} = \mathcal{X}_{\text{free}}$ ,  $\text{cost}(\pi^*(x, y)) = \text{cost}(x, y)$ . Thus, for a sufficiently large  $i$ ,  $\forall x, y \in V_i(S)$  and  $\forall \lambda > 0$ , w.h.p.  $\text{cost}_{\mathcal{H}_i(S)}(x, y) \leq (1 + \lambda)\text{cost}(x, y)$ .  $\square$

**Lemma 6** *Let  $B(x, r)$  denote the  $d$ -dimensional ball centered at  $x$  with radius  $r$ . If  $\mathcal{X} = \mathcal{X}_{\text{free}}$ , then w.h.p. for any  $n$  independent samples chosen uniformly at random in  $\mathcal{X}_{\text{free}}$ , the number of samples located in  $B(x, r)$  is at least  $\frac{\zeta_d \cdot n}{2} r^d$ .*

*Proof.* Consider the  $n$  samples as a sequence of Bernoulli trials, where a successful sample is one that falls into the ball. This happens with probability  $p = \frac{\mu(B(x, r) \cap \mathcal{X}_{\text{free}})}{\mu(\mathcal{X}_{\text{free}})}$ . As  $\mathcal{X}_{\text{free}} = \mathcal{X}$ ,  $\mu(B(x, r)) = \zeta_d \cdot r^d$  and the samples are drawn uniformly from the unit cube,  $p = \zeta_d \cdot r^d$ . Let  $X$  be a random variable that counts the number of samples that fall into the ball. Hoeffding's inequality (see, e.g., [29]) states that

$$\Pr[X < k] \leq \exp\left(-2 \frac{(np - k)^2}{n}\right).$$

Choosing  $k = \frac{np}{2}$  we have that

$$\Pr\left[X < \frac{np}{2}\right] \leq \exp\left(-\frac{np^2}{4}\right),$$

which tends to zero as  $n \rightarrow \infty$ .  $\square$

Let  $l = \text{cost}(x_{\text{init}}, x_{\text{goal}})$  denote the length of the optimal path between the start and the goal configurations. The following lemmas bound  $X_{\text{NN},i}(\text{MPLB}(S))$  from above and bound  $X_{\text{NN},i}(\text{aFMT}^*(S))$  from below w.h.p.

**Lemma 7** *For a sufficiently large  $i$ , w.h.p.,  $X_{\text{NN},i}(\text{aFMT}(S)^*) \geq \frac{\zeta_d \cdot |V_i(S)|}{2} \cdot \left(\frac{l}{1+\lambda}\right)^d$ .*

*Proof.* Note that assuming that the following conditions hold **(C1)**  $\text{cost}_{\mathcal{H}_i(S)}(x, y) \leq (1 + \lambda)\text{cost}(x, y)$ , and **(C2)** given  $n$  collision-free samples, the number of samples located in  $B(x, r)$  is at least  $\frac{\zeta_d \cdot n}{2} r^d$ , and due to the fact that  $c_i(\text{aFMT}^*) \geq l$ ,

then

$$\begin{aligned}
|V_i(\text{aFMT}^*)| &= \left| \left\{ x \in V_i(S) \mid \text{cost-to-come}_{\mathcal{H}_i(S)}(x) \leq c_i(\text{aFMT}^*) \right\} \right| \\
&\geq \left| \left\{ x \in V_i(S) \mid \text{cost}(x_{init}, x) \leq \frac{c_i(\text{aFMT}^*)}{1+\lambda} \right\} \right| \\
&\geq \frac{\zeta_d \cdot |V_i(S)|}{2} \cdot \left( \frac{c_i(\text{aFMT}^*)}{1+\lambda} \right)^d \\
&\geq \frac{\zeta_d \cdot |V_i(S)|}{2} \cdot \left( \frac{l}{1+\lambda} \right)^d.
\end{aligned}$$

Now, recall that Observation 2 bounds the number of NN calls performed by aFMT\* for any random sequence of samples  $S$ . Namely, for any random sequence  $S$ ,  $X_{\text{NN},i}(\text{aFMT}^*) \geq |V_i(\text{aFMT}^*)|$ . Thus,

$$\Pr \left[ X_{\text{NN},i}(\text{aFMT}^*) \geq \frac{\zeta_d \cdot |V_i(S)|}{2} \cdot \left( \frac{l}{1+\lambda} \right)^d \right] \geq \Pr[\text{Conditions (C1) and (C2) hold}].$$

By Lemma 5 and Lemma 6 for a sufficiently large  $i$  the probability that both conditions (C1) and (C2) will hold will occur w.h.p.  $\square$

**Lemma 8** *For a sufficiently large  $i$ , w.h.p.,  $X_{\text{NN},i}(\text{MPLB}(S)) \leq 2 \frac{\zeta_d \cdot |V_i(S)|}{2} \cdot \left( \frac{(1+\lambda)l}{2} \right)^d$ .*

*Proof.* Recall that Observation 3 quantifies the number of NN calls for the MPLB algorithm as,

$$\begin{aligned}
\#_{\text{NN},i}(\text{MPLB}) &\leq \left| \left\{ x \in V_i(S) \mid \text{cost-to-come}_{\mathcal{G}_i(S)}(x) \leq \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)} \right\} \right| \\
&\quad + \left| \left\{ x \in V_i(S) \mid \text{cost-to-go}_{\mathcal{G}_i(S)}(x) \leq \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)} \right\} \right|.
\end{aligned}$$

Using Observation 3 and due to the fact that  $c_i(\text{aFMT}^*) \geq l$  and that  $\text{cost}(x, y) \leq \text{cost}_{\mathcal{H}_i(S)}(x, y)$  and assuming that condition **(C2)** (stated in Lemma 7) holds, then

$$\begin{aligned}
X_{\text{NN},i}(\text{MPLB}(S)) &\leq \left| \left\{ x \in V_i(S) \mid \text{cost}(x_{init}, x) \leq \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)} \right\} \right| \\
&\quad + \left| \left\{ x \in V_i(S) \mid \text{cost}(x_{goal}, x) \leq \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)} \right\} \right| \\
&\leq 2 \frac{\zeta_d \cdot |V_i(S)|}{2} \cdot \left( \frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)} \right)^d \\
&\leq 2 \frac{\zeta_d \cdot |V_i(S)|}{2} \cdot \left( \frac{(1+\lambda)l}{2} \right)^d.
\end{aligned}$$

Thus,

$$\Pr \left[ X_{\text{NN},i}(\text{MPLB}(S)) \leq 2 \frac{\zeta_d \cdot |V_i(S)|}{2} \cdot \left( \frac{(1+\lambda)l}{2} \right)^d \right] \geq \Pr[\text{Condition (C2) holds}].$$

By Lemma 6 for a sufficiently large  $i$  the probability that condition (C2) holds will occur w.h.p.  $\square$

Thus, by Lemmas 7 and 8, w.h.p. for any  $\lambda < (\sqrt{2})^{\frac{d-1}{d}} - 1$ ,  $X_{\text{NN}}(\text{MPLB}(S)) < X_{\text{NN}}(\text{aFMT}^*(S))$ , which completes the proof of Theorem 1.  $\square$

The result above should not come as a surprise. Clearly, the cost of the path obtained by both algorithms approaches  $l$  as the number of samples grows. As MPLB confines its search to all nodes around  $x_{\text{init}}$  and all nodes around  $x_{\text{goal}}$  with distance less than  $\frac{c_{i-1}(\text{MPLB})}{2(1+\varepsilon)}$ , the volume searched approaches two balls of radius  $\frac{l}{2}$ . aFMT\*, on the other hand, confines its search to all nodes around  $x_{\text{init}}$  with distance less than  $c_i(\text{aFMT}^*)$ , the volume searched approaches one ball of radius  $l$ . Indeed, w.h.p.,  $\lim_{\lambda \rightarrow 0} \frac{X_{\text{NN}}(\text{MPLB}(S))}{X_{\text{NN}}(\text{aFMT}^*(S))} \leq \lim_{\lambda \rightarrow 0} \frac{(1+\lambda)^{2d}}{2^{d-1}} = \frac{1}{2^{d-1}}$ , which is exactly the ratio between two  $d$ -dimensional balls with radius  $\frac{r}{2}$  and one  $d$ -dimensional ball with radius  $r$ .

### 5.3 Well-behaved C-spaces

**Definition 2.** A C-space is said to be  $\xi$ -**approximable** if for every two collision-free configurations  $x, y$  there exists a collision-free path  $P(x, y) \subseteq \mathcal{X}_{\text{free}}$  such that  $|P(x, y)| \leq (1 + \xi) \cdot \text{cost}(x, y)$ .

We give several examples of non trivial  $\xi$ -approximable C-spaces. The first, is a C-space whose C-obstacles are a set of disjoint balls. It is easy to see that such a C-space is  $\frac{\pi}{2}$ -approximable. The second example uses the notion of  $\alpha$ -fatness used in computational geometry (see e.g. [5, 27]): An object is said to be  $\alpha$ -fat if every ball centered in the object that intersects its boundary has at least  $\alpha$  of its volume inside the object. Chew et al. [7] showed that there exists a path between any two points on the boundary of a convex fat obstacle whose length is at most  $\xi$  times the Euclidean distance between the points. Here  $\xi$  is a constant that depends only on the dimension  $d$  and on the fatness parameter  $\alpha$ . More precisely,  $\xi = 1 + \frac{4}{\pi\alpha}$  for  $d = 2$ , and  $\xi = 1 + \frac{8d^d}{\alpha}$  for  $d > 2$ .

**Definition 3.** A C-space is said to be  $\gamma$ -**spaced** for  $\gamma \in (0, 1]$  if for every point  $x \in \mathcal{X}_{\text{free}}$  and for every  $r \in \mathbb{R}$  the ball centered at  $x$  with radius  $r$  has at least  $\gamma$  of its volume contained within  $\mathcal{X}_{\text{free}}$ , namely,  $\mu(B(x, r) \cap \mathcal{X}_{\text{free}}) \geq \gamma \cdot \mu(B(x, r))$ .

**Definition 4.** A C-space is said to be  $(\xi, \gamma)$ -**well behaved** if it is  $\xi$ -approximable and  $\gamma$ -spaced.

For a  $(\xi, \gamma)$ -well behaved C-space, we can generalize Lemma 5 and Lemma 6:

**Lemma 9** *For a  $(\xi, \gamma)$ -well behaved C-space and for a sufficiently large  $i$ ,  $\forall x, y \in V_i(S)$  and  $\forall \lambda > 0$ , w.h.p.,  $\text{cost}(x, y) \leq \text{cost}_{\mathcal{H}_i(S)}(x, y) \leq (1 + \lambda) \cdot \xi \cdot \text{cost}(x, y)$ .*

**Lemma 10** *For a  $(\xi, \gamma)$ -well behaved C-space, given any  $n$  independent samples chosen uniformly at random in  $\mathcal{X}_{\text{free}}$ , w.h.p. the number of samples located in  $B(x, r)$  is at least  $\frac{\xi_d \cdot n}{2\mu(\mathcal{X}_{\text{free}})} \gamma r^d$ .*

Thus, using Lemma 9 and Lemma 10 and the same arguments for the case of no obstacles, we have that:

**Corollary 2** *For a  $(\xi, \gamma)$ -well behaved C-space,*

$$\lim_{\lambda \rightarrow 0} \frac{X_{\text{NN}}(\text{MPLB}(S))}{X_{\text{NN}}(\text{aFMT}^*(S))} \leq \lim_{\lambda \rightarrow 0} \frac{(1 + \lambda)^{2d} \xi^{2d}}{\gamma^{2d-1}} = \frac{\xi^{2d}}{\gamma^{2d-1}}.$$

This gives w.h.p. an upper bound on the additional number of NN calls performed by MPLB when compared to aFMT\* for  $(\xi, \gamma)$ -well behaved C-spaces.

Specifically, for a C-space with obstacles that are balls which is  $\gamma$ -spaced for some  $\gamma > 0$  we have that w.h.p.:

$$\lim_{\lambda \rightarrow 0} \frac{X_{\text{NN}}(\text{MPLB}(S))}{X_{\text{NN}}(\text{aFMT}^*(S))} \leq \frac{\pi^{2d}}{\gamma^{2d-1}}.$$

For a C-space with  $\alpha$ -fat, convex obstacles which is  $\gamma$ -spaced for some  $\gamma, \alpha > 0$  we have that w.h.p. for  $d = 2$ :

$$\lim_{\lambda \rightarrow 0} \frac{X_{\text{NN}}(\text{MPLB}(S))}{X_{\text{NN}}(\text{aFMT}^*(S))} \leq \frac{(1 + \frac{4}{\pi\alpha})^4}{2\gamma},$$

and for  $d > 2$  w.h.p.

$$\lim_{\lambda \rightarrow 0} \frac{X_{\text{NN}}(\text{MPLB}(S))}{X_{\text{NN}}(\text{aFMT}^*(S))} \leq \frac{(1 + \frac{8d^d}{\alpha})^{2d}}{\gamma^{2d-1}}.$$

## 6 Evaluation

We present an experimental evaluation of the performance of MPLB as an anytime algorithm on different scenarios consisting of 3 and 6 DoFs. All experiments were run using the Open Motion Planning Library (OMPL 0.10.2) [9] on a 3.4GHz Intel Core i7 processor with 8GB of memory. The implementation of aFMT\* and MPLB is based on the implementation of FMT\* as provided by Pavone's research group. All results are averaged over ten different runs. The Corridors scenario (Figure 1a) and the Alternating barriers Scenario (Figure 2a) are available at <http://acg.cs.tau.ac.il/projects/MPLB/project-page>.



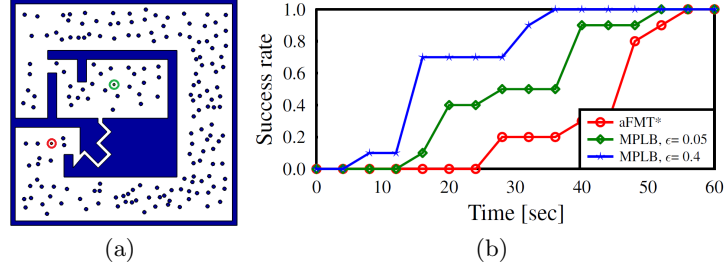


Fig. 1: (a) Corridors scenario—Start and target configurations are depicted by green and red circles, respectively. (b) Success rate to find a path through the narrow corridor.

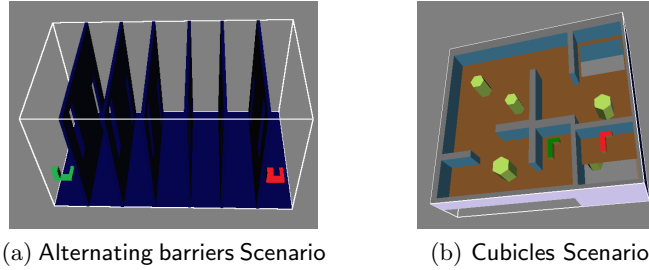


Fig. 2: Scenarios with 6 DoFs. Robot’s start and target configurations are depicted in green and red. The Cubicles scenario is provided with the OMPL [9] distribution.

### 6.1 Fast convergence

We start by comparing the aFMT\* and MPLB algorithms for the case of a three-dimensional C-space which we call the Corridors scenario (Figure 1a). It consists of a planar hexagon robot that can translate and rotate amidst a collection of small obstacles. There are two main corridors from the start to the goal position, a wide one and a narrow one. A relatively small number of samples suffices to find a path through the wide corridor, yet in order to compute a low-cost path through the narrow corridor, much more samples are needed. This demonstrates how MPLB refrains from refining an existing solution when no significant advantage can be obtained. Figure 1b depicts the success rate of finding a path through the narrow corridor as time progress. Clearly, for these types of settings MPLB performs favorably over aFMT\* even for large approximation factors. For example, to reach a 70% success rate in finding a path through the narrow corridor, MPLB, run with  $\epsilon = 0.4$ , needs a third of the time needed by aFMT\*.

### 6.2 Nearest Neighbors and Local Planning calls

We continue comparing aFMT\* and MPLB on settings with 6 DoFs. The Alternating barriers scenario (Figure 2a) consists of a robot with three perpendicular

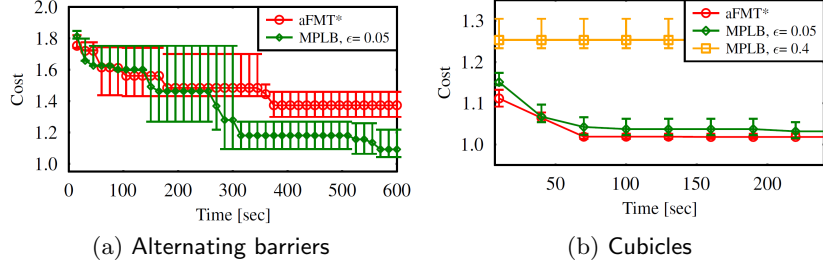


Fig. 3: Average cost vs. time. Cost values are given as a factor of the optimal cost where a cost of one represents the cost of an optimal path. Low and high error bars denote the twentieth and eightieth percentile, respectively.

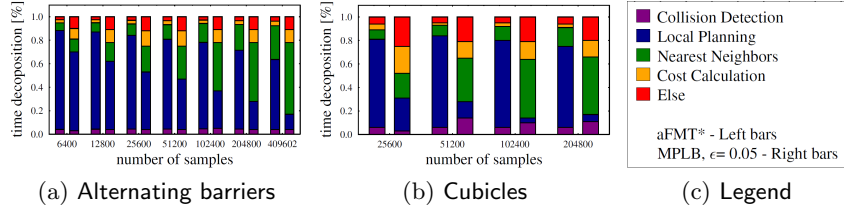


Fig. 4: Percentage of time spent for each of the main components in each iteration for both algorithms. Each iteration is represented by the number of samples used, the bars on the left (right) of each iteration represent the result of aFMT\* (MPLB, respectively). Note that the time of each iteration for each algorithm is different, the results are presented in percentages of the total running time for each algorithm.

rods free-flying in space. The robot needs to pass through a series of barriers each containing a large and a small hole. The large holes are located on alternating sides of consecutive barriers. Thus, an easy path to find would be to cross each barrier through a large hole. A short path would require passing through a small hole after each large hole. We first report on the average cost computed vs. time for aFMT\* and for MPLB run with  $\epsilon = 0.05$ . The results, depicted in Figure 3a, show that MPLB converges faster to a lower-cost solution than aFMT\*. It should be noted that the error bars for MPLB are larger than that of aFMT\* meaning that there is a larger variance in the results. Perhaps more interesting is the analysis of the running time with regards to the main components used by the algorithms. We profiled both algorithms and collected the total time spent on CD for points, LP for edges, NN calls and cost computations. Figure 4a depicts the percentage of time spent for each of the main components in each iteration for both algorithms. The left bar for each iteration represents the aFMT\* algorithm while the right bar represents the MPLB algorithm. Clearly, CD computation time (due to sampling, not LP) is negligible for both algorithms and cost calculation plays a larger (but still small) role in the MPLB algorithm. CD calls due to LP calls are the main bottleneck for aFMT\* (starting at around 80% and gradually decreasing to 60%) while for MPLB they start as a main time

consumer but as samples are added their percentage of the overall iteration time becomes quite small (around 15% for the last iteration). NN calls play an almost complementary role to the LP and for the last iteration take 60% of the total running time for MPLB algorithm (while taking less than 30% for aFMT\*).

We performed the same experiment for the Cubicles scenario which consists of an L-shaped robots free-flying in space amidst a sparse collection of obstacles<sup>5</sup>. Figure 2b depicts the scenario, Figure 3b the average cost and Figure 4b the percentage of time spent for each of the main components in each iteration for both algorithms. Although here the cost produced by aFMT\* is slightly smaller than that of MPLB, it is of course within the approximation factor provided. The percentage of time spent for LP and NN is similar to the results presented for the Alternating Barriers scenario.

The table to the right reports the ratio of the number of NN and LP calls between MPLB and aFMT\* for the Cubicles scenario. One can see that, indeed, the number of NN calls that MPLB performs is higher than aFMT\* but the excess is nowhere close to the conservative upper bounds we give in Section 5. Additionally, as expected, the number of LP calls is much smaller for MPLB than for aFMT\*.

$n$	the ratio $\frac{\#_{\text{NN}}(\text{MPLB})}{\#_{\text{NN}}(\text{aFMT}^*)}$	the ratio $\frac{\#_{\text{LP}}(\text{MPLB})}{\#_{\text{LP}}(\text{aFMT}^*)}$
6,400	1.2	0.98
12,800	1.47	0.99
25,600	2.42	0.61
51,200	1	0.07
102,400	1.29	0.03
204,800	1.01	0.04

## 7 Conclusion and outlook

In this work we show that by using effective lower bounds and with little or no compromise on the cost of paths produced by the algorithm, the weight of CD (via LP calls) becomes almost negligible with respect to NN calls. This follows the ideas presented by Bialkowski et al [6] but uses different, more general, methods. Looking into NN computation, one can notice that several AO algorithms (e.g., sPRM\* [16], FMT\*, aFMT\*) rely on a specific type of NN computation: given a set of  $n$  points, either compute for each point all its  $k$  nearest neighbors, or compute for each point all points within distance  $r$  from it. In both cases, the set of points are known in advance (i.e. there are no “general” queries to this NN problem) and  $k$  (or  $r$ ) are parameters that do not change throughout the algorithm or throughout a single iteration of the algorithm.

This calls for using application-specific NN algorithms and not necessarily general purpose ones (such as  $kd$ -trees [11]). Such algorithms exist, for example, in high dimensions locally sensitive hashing (LSH) [13] may be used. For smaller dimensions, the work by Aiger et al. [1] may be a good, practical choice.

Directions for further research include giving tighter bounds on the number of NN calls used by MPLB: We presented bounds on the number of calls for well-behaved spaces, yet we believe that these bounds are far from tight and we wish to generalize the result to other  $C$ -spaces as well.

<sup>5</sup> The Cubicles Scenario is provided as part of the OMPL distribution.

## 8 Acknowledgements

We wish to thank Marco Pavone and his co-workers for their advice and support regarding the FMT\* algorithm. Additionally we wish to thank Kiril Solovey for fruitful discussions and insightful comments regarding this work.

## A Visualization of aFMT\* and MPLB

To further clarify the behavior of aFMT\* and MPLB we consider a point robot translating in the plane. We run both algorithms for four iterations using  $50 * 2^i$  samples in the  $i$ 'th iteration. The setting contains one obstacle within a square room and source and target configurations located at the upper left and lower left corners of the square, respectively. Bypassing the obstacle in a clockwise fashion is easy as there are paths with high clearance while bypassing the obstacle in a counterclockwise fashion is much harder due to the low clearance along such paths.

Figure 5 depicts the disk graph  $\mathcal{G}_i$  for each iteration, the search-tree of aFMT\* and the search-tree of MPLB.

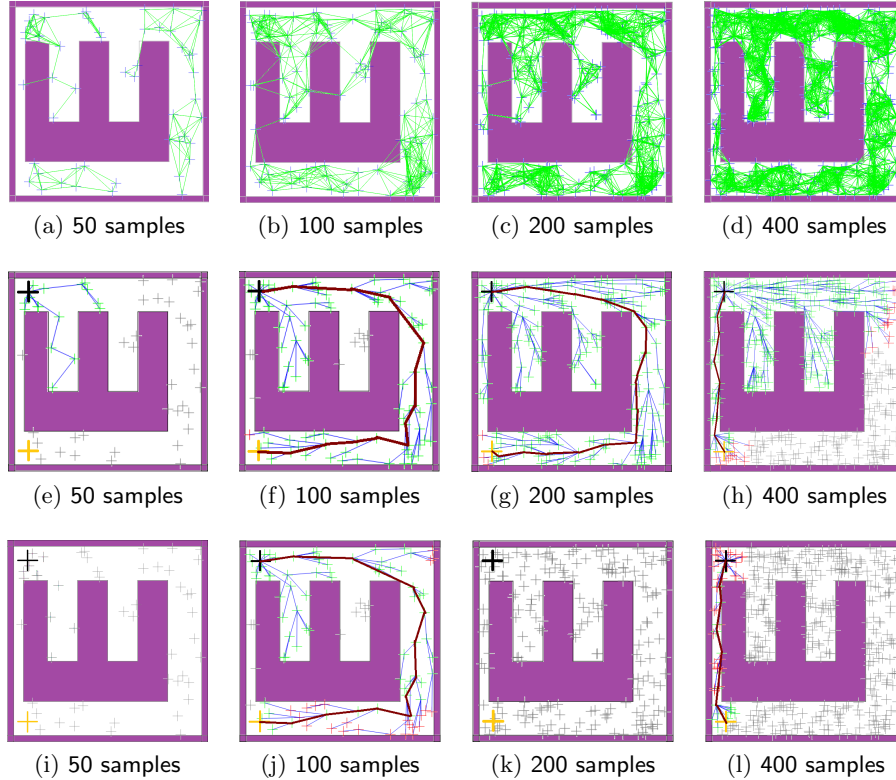


Fig. 5: (a)-(d) the disk graph  $\mathcal{G}$ , (e)-(h) aFMT\* search-tree using  $\mathcal{H}$  and (i)-(l) MPLB search tree with  $\varepsilon = 0.2$  using  $\mathcal{H}$ .

## B Lower Bound Tree FMT\*

We suggest to apply the concepts suggested in the work on LBT-RRT [26] to the FMT\* algorithm and note that this can be applied both to aFMT\* and to MPLB. For ease of exposition we describe the tool applied to aFMT\* (see Algorithm 2) and call this variant LBT-FMT\*. The application to MPLB is then a fairly straightforward extension. We suggest to compute an *asymptotically near-optimal* solution by constructing two trees simultaneously. The first tree  $\mathcal{T}_{lb}$  stores a *lower bound* on the cost to reach each node while the second tree,  $\mathcal{T}_{apx}$  stores an approximation of the cost to reach each node. At every stage of the algorithm we will maintain the following invariants:

1. **Bounded approximation invariant**<sup>6</sup>

For each node  $x \in \mathcal{T}_{lb}, \mathcal{T}_{apx}$ ,  $\text{cost-to-go}_{\mathcal{T}_{apx}}(x) \leq (1 + \varepsilon) \cdot \text{cost-to-go}_{\mathcal{T}_{lb}}(x)$ .

2. **Lower bound invariant**

For each node  $x \in \mathcal{T}_{lb}$ ,  $\text{cost-to-go}_{\mathcal{T}_{lb}}(x) \leq \text{cost-to-go}_{\text{FMT}^*}(x)$  after  $x$  is in the tree of the FMT\* algorithm, which was run on the same set of nodes.

The Bounded approximation invariant is maintained by construction (using a proof similar to the one presented in [26]). Let us note several straightforward (yet helpful) observations:

**Observation 4** *The set of nodes of  $\mathcal{T}_{lb}$  and  $\mathcal{T}_{apx}$  are identical at every stage of the algorithm.*

**Observation 5** *Every edge of  $\mathcal{T}_{apx}$  is collision free.*

Now,

**Lemma 11** *The Lower bound invariant is maintained by LBT-FMT\**

*Proof.* The proof is by induction on the nodes in  $H$  where the base trivially holds for  $x_{init}$ . Now, for the induction step, assume that for every node in  $H$  the lower bound invariant holds. Let  $z$  be the node with the lowest cost in  $H$  (i.e., the node with the lowest cost in  $\mathcal{T}_{lb}$ ). As the induction hypothesis holds we know that  $\text{cost-to-go}_{\mathcal{T}_{lb}}(z) \leq \text{cost-to-go}_{\text{FMT}^*}(z)$  after  $z$  is in the tree of the FMT\* algorithm run on the same set of nodes.

Recall that  $X_{near} \leftarrow W \cap N_z$  is the set of nearest vertices of  $z$  not in the search tree and that for each such  $x$ ,  $Y_{near} \leftarrow H \cap N_x$  is the set of nearest vertices of  $x$  in the search tree. We show for every  $x \in X_{near}$  that if it is added to  $H$  then the lower bound invariant is maintained for  $x$  namely,  $\text{cost-to-go}_{\mathcal{T}_{lb}}(x) \leq \text{cost-to-go}_{\text{FMT}^*}(x)$ .

If  $x$  is added to  $H$  then

$$\text{cost-to-go}_{\mathcal{T}_{lb}}(x) = \text{cost-to-go}_{\mathcal{T}_{lb}}(y_{min\ lb}) + \text{cost}(y_{min\ lb}, x)$$

---

<sup>6</sup> The bounded approximation invariant presented in the appendix is not to be confused with the bounded approximation invariant used for the MPLB algorithm. We use the same name to be consistent with [26].

---

**Algorithm 2** LBT-FMT\* (Cache)

---

```

1:  $V \leftarrow \{x_{init}\} \cup \text{sample\_free}(n); \quad E \leftarrow \emptyset$ 
2:  $W \leftarrow V \setminus \{x_{init}\}; \quad H \leftarrow \{x_{init}\} \quad // H \text{ is ordered according to } \text{cost}_{\mathcal{T}_{lb}}$ 
3: for all  $v \in V$  do
4:    $N_v \leftarrow \text{nearest\_neighbors}(V \setminus \{v\}, v, r_n)$ 
5:  $z \leftarrow x_{init}$ 
6: while  $z \notin \mathcal{X}_{Goal}$  do
7:    $H_{new} \leftarrow \emptyset$ 
8:    $X_{near} \leftarrow W \cap N_z \quad // \text{nearest vertices of } z \text{ not in roadmap}$ 
9:   for  $x \in X_{near}$  do
10:     $Y_{near} \leftarrow H \cap N_x \quad // \text{nearest vertices of } x \text{ in roadmap}$ 
11:     $y_{min \text{ lb}} \leftarrow \arg \min_{y \in Y_{near}} \{\text{cost-to-go}_{\mathcal{T}_{lb}}(y) + \text{cost}(y, x)\}$ 
12:     $y_{min \text{ apx}} \leftarrow \arg \min_{y \in Y_{near}} \{\text{cost-to-go}_{\mathcal{T}_{apx}}(y) + \text{cost}(y, x) \quad \text{and}$ 
       $\text{collision\_free\_using\_cache}(y, x)\}$ 
13:    if  $\text{cost-to-go}_{\mathcal{T}_{apx}}(y_{min \text{ apx}}) + \text{cost}(y_{min \text{ apx}}, x) \leq (1 + \varepsilon) \cdot$ 
       $(\text{cost-to-go}_{\mathcal{T}_{lb}}(y_{min \text{ lb}}) + \text{cost}(y_{min \text{ lb}}, x))$  then
14:       $\mathcal{T}_{lb}.\text{parent}(x) \leftarrow y_{min \text{ lb}}$ 
15:       $\mathcal{T}_{apx}.\text{parent}(x) \leftarrow y_{min \text{ apx}}$ 
16:       $H_{new} \leftarrow H_{new} \cup \{x\} \quad W \leftarrow W \setminus \{x\}$ 
17:    else
18:      if  $\text{collision\_free}(y_{min \text{ lb}}, x)$  then
19:         $\mathcal{T}_{lb}.\text{parent}(x) \leftarrow y_{min \text{ lb}}$ 
20:         $\mathcal{T}_{apx}.\text{parent}(x) \leftarrow y_{min \text{ lb}}$ 
21:         $H_{new} \leftarrow H_{new} \cup \{x\} \quad W \leftarrow W \setminus \{x\}$ 
22:    $H \leftarrow (H \cup H_{new}) \setminus \{z\}$ 
23:   if  $H = \emptyset$  then
24:     return FAILURE
25:    $z \leftarrow \arg \min_{y \in H} \{\text{cost}(y)\}$ 
26: return PATH

```

---

where

$$y_{min \text{ lb}} = \arg \min_{y \in Y_{near}} \{\text{cost-to-go}_{\mathcal{T}_{lb}}(y) + \text{cost}(y, x)\}.$$

There exists a node  $z' \in Y_{near}$  such that  $\text{cost-to-go}_{\mathcal{T}_{lb}}(z) \leq \text{cost-to-go}_{\mathcal{T}_{lb}}(z')$  and  $\text{cost-to-go}_{\text{FMT}^*}(x) = \text{cost-to-go}_{\text{FMT}^*}(z') + \text{cost}(z', x)$ .  
Now,

$$\begin{aligned}
\text{cost-to-go}_{\mathcal{T}_{lb}}(x) &= \text{cost-to-go}_{\mathcal{T}_{lb}}(y_{min \text{ lb}}) + \text{cost}(y_{min \text{ lb}}, x) \\
&\leq \text{cost-to-go}_{\mathcal{T}_{lb}}(z') + \text{cost}(z', x) \\
&\leq \text{cost-to-go}_{\text{FMT}^*}(z') + \text{cost}(z', x) \\
&= \text{cost-to-go}_{\text{FMT}^*}(x).
\end{aligned}$$

Where the first equality is by the behavior of the algorithm (line 14 or line 19), the first inequality comes from the fact that  $y_{min\ lb}$  attains the minimum over  $Y_{near}$  and the second inequality comes from the induction hypothesis.

Note that this technique is general for any cached information maintained by the algorithm. Specifically, we use the cache provided by previous iterations.

## References

- [1] Aiger, D., Kaplan, H., Sharir, M.: Reporting neighbors in high-dimensional Euclidean spaces. In: SODA. pp. 784–803 (2013)
- [2] Akgun, B., Stilman, M.: Sampling heuristics for optimal motion planning in high dimensions. In: IROS. pp. 2640–2645 (2011)
- [3] Alterovitz, R., Patil, S., Derbakova, A.: Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In: ICRA. pp. 3706–3712 (2011)
- [4] Arslan, O., Tsiotras, P.: Use of relaxation methods in sampling-based algorithms for optimal motion planning. In: ICRA. pp. 2421–2428 (2013)
- [5] de Berg, M., van der Stappen, F., Vleugels, J., Katz, M.: Realistic input models for geometric algorithms. *Algorithmica* 34(1), 81–97 (2002)
- [6] Bialkowski, J., Karaman, S., Otte, M., Frazzoli, E.: Efficient collision checking in sampling-based motion planning. In: WAFR. pp. 365–380 (2012)
- [7] Chew, L.P., David, H., Katz, M.J., Kedem, K.: Walking around fat obstacles. *Inf. Process. Lett.* 83(3), 135–140 (2002)
- [8] Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., Thrun, S.: *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press (June 2005)
- [9] Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* (2012)
- [10] Dobson, A., Bekris, K.E.: Improving sparse roadmap spanners. In: ICRA. pp. 4106–4111 (2013)
- [11] Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.* 3(3), 209–226 (Sep 1977)
- [12] Hsu, D., Latombe, J.C., Motwani, R.: Path planning in expansive configuration spaces. *Int. J. Comput. Geometry Appl.* 9(4/5), 495–512 (1999)
- [13] Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: *ACM symposium on Theory of computing*. pp. 604–613. ACM (1998)
- [14] Islam, F., Nasir, J., Malik, U., Ayaz, Y., Hasan, O.: RRT\*-Smart: Rapid convergence implementation of RRT\* towards optimal solution. In: *Intl Conf on Mechatronics and Automation*. pp. 1651–1656 (2013)
- [15] Janson, L., Pavone, M.: Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions. *CoRR* abs/1306.3532 (2013)
- [16] Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. *I. J. Robot. Res.* 30(7), 846–894 (2011)
- [17] Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Trans. Robot.* 12(4), 566–580 (1996)

- [18] Koenig, S., Likhachev, M., Furcy, D.: Lifelong planning A\*. *Artif. Intell.* 155(1-2), 93–146 (2004)
- [19] Kuffner, J.J.: Effective sampling and distance metrics for 3d rigid body path planning. In: ICRA. pp. 3993–3998 (2004)
- [20] Kuffner, J.J., LaValle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: ICRA. pp. 995–1001 (2000)
- [21] LaValle, S.M.: *Planning algorithms*. Cambridge University Press (2006)
- [22] Littlefield, Z., Li, Y., Bekris, K.E.: Efficient sampling-based motion planning with asymptotic near-optimality guarantees for systems with dynamics. In: IROS. pp. 1779–1785 (2013)
- [23] Marble, J.D., Bekris, K.E.: Towards small asymptotically near-optimal roadmaps. In: ICRA. pp. 2557–2562 (2012)
- [24] Pearl, J.: *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley (1984)
- [25] Reif, J.H.: Complexity of the mover’s problem and generalizations (extended abstract). In: FOCS. pp. 421–427 (1979)
- [26] Salzman, O., Halperin, D.: Asymptotically near-optimal RRT for fast, high-quality, motion planning. In: ICRA (2014), to appear
- [27] van der Stappen, A.F., Halperin, D., Overmars, M.H.: The complexity of the free space for a robot moving amidst fat obstacles. *Comput. Geom.* 3, 353–373 (1993)
- [28] Wang, W., Balkcom, D., Chakrabarti, A.: A fast streaming spanner algorithm for incrementally constructing sparse roadmaps. IROS (2013)
- [29] Wikipedia: Binomial distribution — wikipedia, the free encyclopedia (2014), [http://en.wikipedia.org/w/index.php?title=Binomial\\_distribution&oldid=598857617](http://en.wikipedia.org/w/index.php?title=Binomial_distribution&oldid=598857617), [Online; accessed 16-March-2014]